

Alternating Anderson-Richardson method: An efficient alternative to preconditioned Krylov methods for large, sparse linear systems

Phanish Suryanarayana^{*,a}, Phanisri P. Pratapa^a, John E. Pask^b

^aCollege of Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA

^bPhysics Division, Lawrence Livermore National Laboratory, Livermore, CA 94550, USA

Abstract

We generalize the recently proposed Alternating Anderson-Jacobi (AAJ) method (Pratapa et al., *J. Comput. Phys.* (2016), 306, 43–54) to include preconditioning, and demonstrate its efficiency and scaling in the solution of large, sparse linear systems on parallel computers. The resulting preconditioned Alternating Anderson-Richardson (AAR) method reduces to the AAJ method for a particular choice of preconditioner. The AAR method employs Anderson extrapolation at periodic intervals within a preconditioned Richardson iteration to accelerate convergence. In this work, we develop a version of the method that is particularly well suited for scalable high-performance computing. In applications to Helmholtz and Poisson equations, we show that the strong and weak parallel scaling of AAR is superior to both Generalized Minimal Residual (GMRES) and Conjugate Gradient (CG) methods, using the same preconditioning, in large-scale parallel calculations employing up to 110,592 computational cores. Moreover, we find that the minimum time to solution for AAR is up to a factor of two smaller than both GMRES and CG. Overall, we find that the AAR method provides an efficient and scalable alternative to preconditioned Krylov solvers for the solution of large, sparse linear systems on high performance computing platforms.

Key words: Linear systems of equations, Parallel computing, Anderson extrapolation, Richardson iteration, Poisson equation, Helmholtz equation

1. Introduction

Linear systems of equations are encountered in the gamut of applications areas, from quantum to continuum to celestial mechanics. For relatively small system sizes, direct methods (e.g., QR decomposition) are generally the preferred solution schemes. However, as the size of the system increases, direct methods become inefficient—in terms of both computational cost and storage requirements—relative to iterative approaches, particularly Krylov subspace based methods such as Generalized Minimal Residual (GMRES) [1] and Conjugate Gradient (CG) [2] methods. Therefore, such iterative approaches are often the methods of choice for the solution of large-scale linear systems of equations.

A number of physical applications require the repeated solution of large, sparse linear systems. For example, in real-space quantum molecular dynamics calculations [3], the Poisson equation may need to be solved hundreds of thousands of times within a single simulation. Therefore, it is critical to reduce the time to solution as far as possible in such situations. This is typically achieved by recourse to parallel computing, whereby the the number of floating point operations that can be performed per second is significantly

^{*}Corresponding Author (phanish.suryanarayana@ce.gatech.edu)

increased. However, the cost associated with inter-processor communication, especially global communication, severely restricts the parallel efficiency of linear solvers, which in turn limits the reduction in wall time that can be achieved in practice. Therefore, there is wide interest in developing algorithms that scale well on modern large-scale parallel computers that routinely contain tens to hundreds of thousands of computational cores or more.

Unfortunately, Krylov subspace methods such as GMRES and CG have limited parallel scalability due to the large number of global operations inherent to them. In this context, the classical Richardson and Jacobi fixed-point iterations [4] are ideally suited for massive parallelization [5, 6, 7] by virtue of the strict locality of operations required. However, they suffer from extremely large prefactors and poor scaling with system size compared to Krylov subspace methods, which has made them unattractive on even the largest modern platforms. This motivates the development of strategies that significantly accelerate the convergence of the basic Jacobi and Richardson iterations, while maintaining their underlying locality and simplicity to the maximum extent possible. With this goal in mind, it has been recently proposed to employ Anderson extrapolation [8] at periodic intervals within the classical Jacobi iteration, resulting in the so called Alternating Anderson-Jacobi (AAJ) method [7]. This strategy has been found to accelerate the Jacobi iteration by multiple orders of magnitude, to the point in fact of outperforming GMRES by up to an order of magnitude in serial computations without preconditioning¹.

In the present work, we generalize the AAJ method to include preconditioning, and demonstrate its efficiency and scaling in large-scale parallel calculations. The resulting preconditioned Alternating Anderson-Richardson (AAR) method reduces to the AAJ method for a particular choice of preconditioner. In applications to Helmholtz and Poisson equations, we show that the strong and weak parallel scaling of AAR is superior to both GMRES and CG, using the same preconditioning, in large-scale parallel calculations employing up to 110,592 computational cores. Moreover, we find that the minimum time to solution for AAR is up to a factor of two smaller than both GMRES and CG.

The remainder of this paper is organized as follows. In Section 2, we describe the preconditioned AAR method. We demonstrate the efficiency and parallel scaling of the method in Section 3. Finally, we conclude in Section 4.

2. Preconditioned Alternating Anderson-Richardson method

In this section, we present the preconditioned Alternating Anderson-Richardson (AAR) method for the solution of large, sparse linear systems

$$\begin{aligned} \mathbf{Ax} &= \mathbf{b}, \\ \mathbf{A} &\in \mathbb{C}^{N \times N}, \quad \mathbf{x} \in \mathbb{C}^{N \times 1}, \quad \mathbf{b} \in \mathbb{C}^{N \times 1}, \end{aligned} \tag{1}$$

where \mathbb{C} is the set of all complex numbers. In this approach, Anderson extrapolation [8] is performed periodically within a preconditioned Richardson fixed-point iteration [4] to accelerate its convergence. Since the method makes no assumptions about the symmetry of \mathbf{A} , it is applicable to symmetric and nonsymmetric systems alike. Moreover, it is amenable to the three types of preconditioning: left, right, and split [4]. For the sake of simplicity, we choose left preconditioning in this work.

¹The analogue of the AAJ method for nonlinear fixed-point iterations is found to accelerate the convergence of the self-consistent field (SCF) method in ab-initio calculations [9].

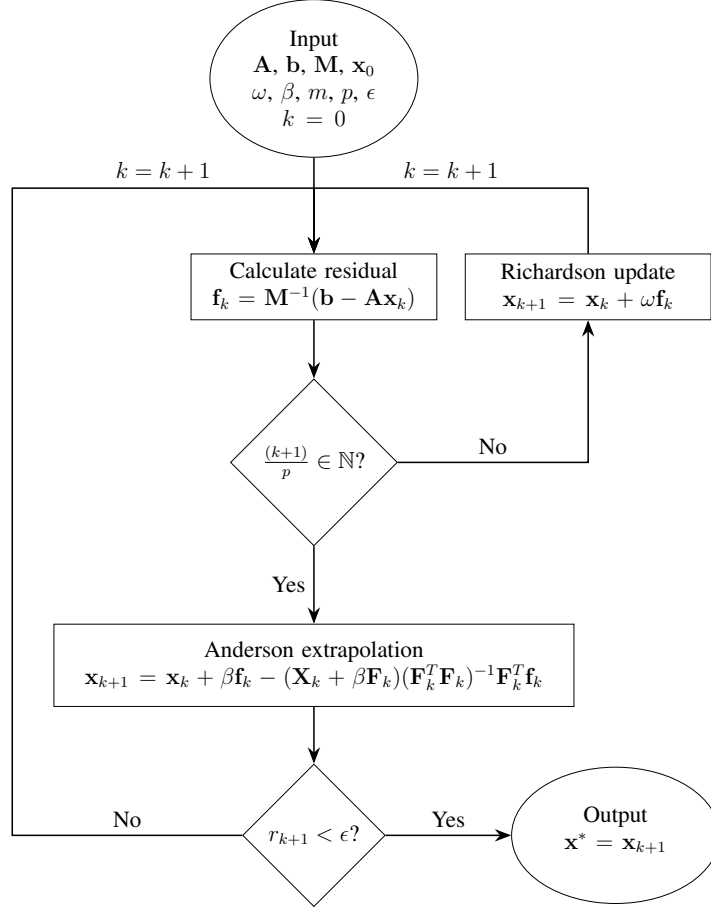


Figure 1: The preconditioned Alternating Anderson-Richardson (AAR) method.

In the AAR method, as in the AAJ method [7], the linear system in Eqn. 1 is solved using a fixed-point iteration of the form

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{B}_k \mathbf{f}_k, \quad (2)$$

where the matrix $\mathbf{B}_k \in \mathbb{C}^{N \times N}$ can be written as

$$\mathbf{B}_k = \begin{cases} \omega \mathbf{I} & \text{if } (k+1)/p \notin \mathbb{N}, \\ \beta \mathbf{I} - (\mathbf{X}_k + \beta \mathbf{F}_k)(\mathbf{F}_k^T \mathbf{F}_k)^{-1} \mathbf{F}_k^T & \text{if } (k+1)/p \in \mathbb{N}. \end{cases} \quad (3)$$

Above, $\omega \in \mathbb{C}$ is the relaxation parameter used in the Richardson update, $\beta \in \mathbb{C}$ is the relaxation parameter used in the Anderson update, $\mathbf{I} \in \mathbb{R}^{N \times N}$ is the identity matrix, the superscript T denotes the conjugate transpose, and p is the frequency of Anderson extrapolation. Additionally, $\mathbf{X}_k \in \mathbb{C}^{N \times m}$ and $\mathbf{F}_k \in \mathbb{C}^{N \times m}$ represent the iteration and residual histories at the k^{th} iteration:

$$\mathbf{X}_k = \begin{bmatrix} (\mathbf{x}_{k-m+1} - \mathbf{x}_{k-m}) & (\mathbf{x}_{k-m+2} - \mathbf{x}_{k-m+1}) & \dots & (\mathbf{x}_k - \mathbf{x}_{k-1}) \end{bmatrix}, \quad (4)$$

$$\mathbf{F}_k = \begin{bmatrix} (\mathbf{f}_{k-m+1} - \mathbf{f}_{k-m}) & (\mathbf{f}_{k-m+2} - \mathbf{f}_{k-m+1}) & \dots & (\mathbf{f}_k - \mathbf{f}_{k-1}) \end{bmatrix}, \quad (5)$$

where $m+1$ is the number of iterates used for Anderson extrapolation, and the residual vector

$$\mathbf{f}_k = \mathbf{M}^{-1}(\mathbf{b} - \mathbf{A}\mathbf{x}_k), \quad (6)$$

with preconditioner $\mathbf{M}^{-1} \in \mathbb{C}^{N \times N}$. The key difference between AAR and AAJ methods lies in the choice of residual vector (Eq. 6). In the AAR method, any available preconditioner \mathbf{M}^{-1} can be employed, whereas in AAJ, $\mathbf{M} = \mathbf{D} = \text{diag}(\mathbf{A})$. The AAR method thus generalizes the AAJ method in the sense that the AAJ method is recovered for the choice of preconditioner $\mathbf{M}^{-1} = \mathbf{D}^{-1}$, i.e., the classical Jacobi preconditioner. We summarize the AAR method in Fig. 1, wherein \mathbf{x}_0 denotes the initial guess, r_k denotes the relative l_2 norm of the residual vector, and ϵ is the tolerance specified for convergence. In order to enhance parallel scalability, we reduce global communications by checking for convergence of the fixed-point iteration (i.e., calculating r_k) only during Anderson extrapolation steps.

We note that just as the AAJ method can be understood as a generalization of Jacobi [4] and Anderson-Jacobi [7] methods, the AAR method can be understood as a generalization of the Richardson [4] and Anderson-Richardson [7] methods. Specifically, the Anderson-Richardson method is recovered for $p = 1$, while the Richardson iteration is obtained in the limit $p \rightarrow \infty$. It is also worth noting that the AAR method with $m \rightarrow \infty$ is equivalent to GMRES without restart, in exact arithmetic. This follows from the fact that the Anderson extrapolation in AAR minimizes the residual over the same Krylov subspace as obtained in GMRES—albeit with a different parameterization. However, in the context of finite-history AAR and restarted GMRES—as employed in practice to reduce orthogonalization and storage costs—AAR is found to outperform GMRES, as demonstrated previously for AAJ in unpreconditioned serial computations [7].

3. Results and discussion

In this section, we demonstrate the efficiency and scaling of the preconditioned Alternating Anderson-Richardson (AAR) method in the parallel solution of large, sparse linear systems of equations. Specifically, we consider the three-dimensional periodic Helmholtz and Poisson equations arising in real-space orbital-free Density Functional Theory (OF-DFT) [10, 11, 12] and Density Functional Theory (DFT) [13, 14, 15, 16] simulations:

$$-\frac{1}{4\pi}\nabla^2 V(\mathbf{r}) + Q V(\mathbf{r}) = P \rho^\alpha(\mathbf{r}) \text{ in } \Omega, \quad \begin{cases} V(\mathbf{r}) = V(\mathbf{r} + L_i \hat{\mathbf{e}}_i) \text{ on } \partial\Omega, \\ \hat{\mathbf{e}}_i \cdot \nabla V(\mathbf{r}) = \hat{\mathbf{e}}_i \cdot \nabla V(\mathbf{r} + L_i \hat{\mathbf{e}}_i) \text{ on } \partial\Omega, \end{cases} \quad (7a)$$

$$-\frac{1}{4\pi}\nabla^2 V(\mathbf{r}) = \rho(\mathbf{r}) + b(\mathbf{r}) \text{ in } \Omega, \quad \begin{cases} V(\mathbf{r}) = V(\mathbf{r} + L_i \hat{\mathbf{e}}_i) \text{ on } \partial\Omega, \\ \hat{\mathbf{e}}_i \cdot \nabla V(\mathbf{r}) = \hat{\mathbf{e}}_i \cdot \nabla V(\mathbf{r} + L_i \hat{\mathbf{e}}_i) \text{ on } \partial\Omega, \end{cases} \quad (7b)$$

where $V(\mathbf{r})$ is the potential, $\rho(\mathbf{r})$ is the electron density, $b(\mathbf{r})$ is the pseudocharge density [17, 18, 19], and Ω is a cuboidal domain with side lengths L_i , unit vectors $\hat{\mathbf{e}}_i$ along each edge, and boundary $\partial\Omega$. In the Helmholtz equation, we set $\alpha = \frac{5}{6} + \frac{\sqrt{5}}{6}$, $P = 0.003277 - i0.009081$, and $Q = -0.134992 - i0.070225$, where $i = \sqrt{-1}$. We discretize the equations using sixth-order accurate finite-differences on a uniform grid with mesh-size h . We perform the computations in parallel by decomposing the domain into cubical subdomains of equal size, with communication between processors handled via Message Passing Interface (MPI). To facilitate parallel implementation, we employ the Portable, Extensible Toolkit for scientific computations (PETSc) [20, 21] suite of data structures and routines, except for the largest calculations (Sec. 3.2), where all operations were implemented in C and MPI directly.

In the Anderson extrapolation step of AAR, we perform only one global communication call (i.e., `MPI_Allreduce`) to simultaneously determine r and the complete matrix $\mathbf{F}_k^T \mathbf{F}_k$. Since the matrix $\mathbf{F}_k^T \mathbf{F}_k$ is generally ill-conditioned, we compute its inverse using the Moore-Penrose pseudoinverse [22]. We choose the common set of parameters $\{\omega, \beta, m, p\} = \{0.6, 0.6, 9, 8\}$ for both the Poisson and Helmholtz equa-

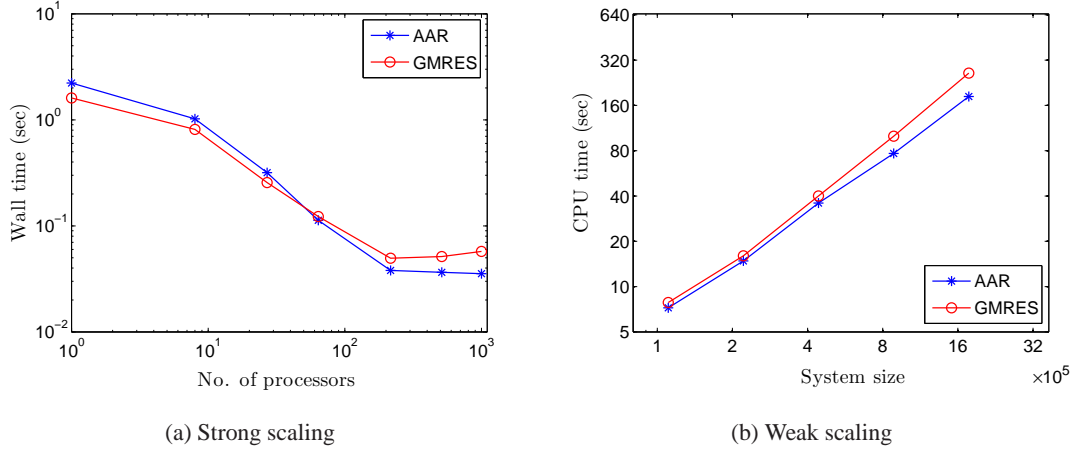


Figure 2: Strong and weak scaling of AAR and GMRES for the periodic Helmholtz equation with ILU(0) block Jacobi preconditioning. In strong scaling, the minimum wall time taken by AAR is 1.30 times smaller than GMRES. In weak scaling, the CPU time taken by AAR and GMRES scales with system size as $\mathcal{O}(N^{1.17})$ and $\mathcal{O}(N^{1.28})$, respectively.

tions.² We compare the performance of AAR with the Conjugate Gradient (CG) [2] and GMRES (restarted every 30 iterations) [1] methods, as implemented in PETSc, except for the largest calculations (Sec. 3.2), where all operations were implemented in C and MPI directly. In all the simulations, we use a vector of all zeros as the starting guess \mathbf{x}_0 and a convergence tolerance of 10^{-6} on the relative residual

$$r_k = \frac{\|\mathbf{A}\mathbf{x}_k - \mathbf{b}\|}{\|\mathbf{b}\|}. \quad (8)$$

In addition, we select the block Jacobi preconditioner [4], since it is particularly suitable for large-scale parallelization. Calculations up to 1,024 cores were carried out on a computer cluster consisting of 16 nodes with the following configuration: Altus 1804i Server - 4P Interlagos Node, Quad AMD Opteron 6276, 16C, 2.3 GHz, 128GB, DDR3-1333 ECC, 80GB SSD, MLC, 2.5" HCA, Mellanox ConnectX 2, 1-port QSFP, QDR, memfree, CentOS, Version 5, and connected through InfiniBand cable. Larger calculations, up to 110,592 cores, were carried out on the Vulcan IBM BG/Q machine at the Lawrence Livermore National Laboratory, consisting of 24,576 compute nodes, with 16 computational cores and 16 GB memory per node, for a total of 393,216 cores and 1.6 PB memory.

3.1. Helmholtz equation

First, we study the performance of AAR relative to GMRES for solving the linear system arising from the discretization of the Helmholtz problem (Eq. 7a). We consider a $3 \times 3 \times 3$ Al supercell based on a face-centered cubic (FCC) unit cell with lattice constant 7.78 Bohr, with atoms randomly displaced from ideal positions. We discretize the domain using a finite-difference grid with a mesh-size of $h = 0.486$ Bohr. For the resulting linear system with ILU(0) block Jacobi preconditioning, we determine the wall time

²Since the performance of AAR is relatively insensitive to the choice of parameters [7], we have not carefully optimized them for the examples studied here.

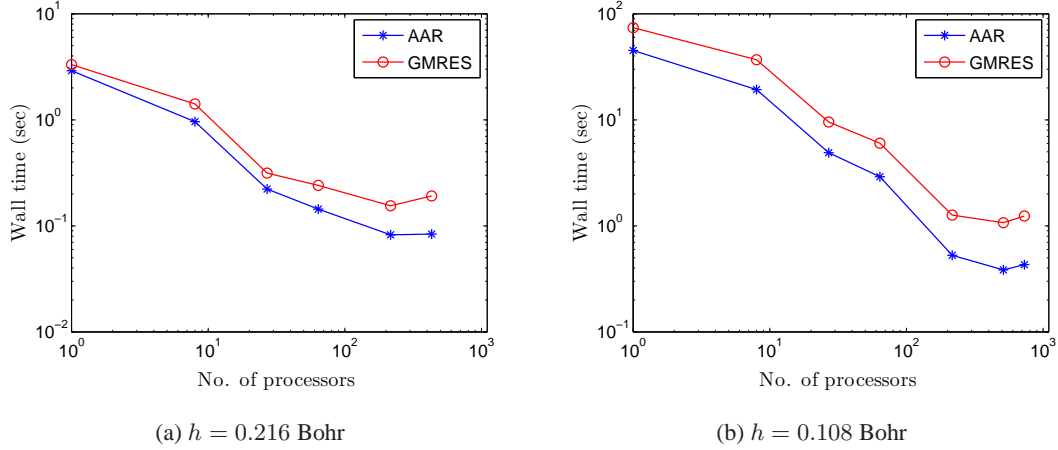


Figure 3: Strong scaling of AAR and GMRES for the periodic Helmholtz equation with Jacobi preconditioning. For $h = 0.216$ Bohr, the minimum wall time taken by AAR is 1.88 times smaller than GMRES. For $h = 0.108$ Bohr, the minimum wall time taken by AAR is 2.79 times smaller than GMRES.

taken by AAR and GMRES on 1, 8, 27, 64, 216, 512, and 1000 cores, and present the results obtained in Fig. 2a. We observe that even though the performances of AAR and GMRES are similar at low core counts (a consequence of both taking similar number of iterations), AAR starts demonstrating superior performance as the core count is increased. In particular, the minimum wall time achieved by AAR is a factor of 1.30 smaller than GMRES due to significantly less global communication in AAR.

We now periodically replicate the above $3 \times 3 \times 3$ system along one direction by factors of 1, 2, 4, 8, and 16, i.e., we generate $3 \times 3 \times 3$, $6 \times 3 \times 3$, $12 \times 3 \times 3$, $24 \times 3 \times 3$, and $48 \times 3 \times 3$ supercells. Correspondingly, we choose 64, 128, 256, 512, and 1024 computational cores. Again, we select $h = 0.486$ Bohr and employ ILU(0) block Jacobi preconditioning. We plot the results of this weak scaling study in Fig. 2b, from which we obtain $\mathcal{O}(N^{1.17})$ and $\mathcal{O}(N^{1.28})$ scaling with system size for AAR and GMRES, respectively. Notably, even though the number of iterations for AAR and GMRES remain constant, both approaches demonstrate slightly superlinear scaling. This is due to the increased cost of global communications at larger core counts. Therefore, the performance of AAR relative to GMRES is expected to further improve as the number of cores increases.

It is worth noting that the performance gap between AAR and GMRES increases as the linear system becomes less well conditioned, as demonstrated in previous work for AAJ in serial computations [7]. In order to verify this result for AAR in parallel computations, we consider the Helmholtz equation for a $1 \times 1 \times 1$ Al supercell with lattice constant of 7.78 Bohr and randomly displaced atoms. To demonstrate the effect of conditioning, we choose a simple Jacobi preconditioner $\mathbf{M}^{-1} = \text{diag}(\mathbf{A})^{-1}$. In Fig. 3, we present the strong scaling of AAR and GMRES for mesh-sizes of $h = 0.216$ and $h = 0.108$ Bohr. We observe that as the mesh gets finer and condition number of \mathbf{A} becomes larger, the speedup of AAR over GMRES increases at both small and large core counts. Specifically, the minimum wall time achieved by AAR is 1.88 and 2.79 times smaller than GMRES for $h = 0.216$ and $h = 0.108$ Bohr, respectively. Therefore, we conclude that as the solution of the linear system becomes more challenging (e.g., in the absence of an effective preconditioner), the speedup of AAR over GMRES is expected to become more substantial in both

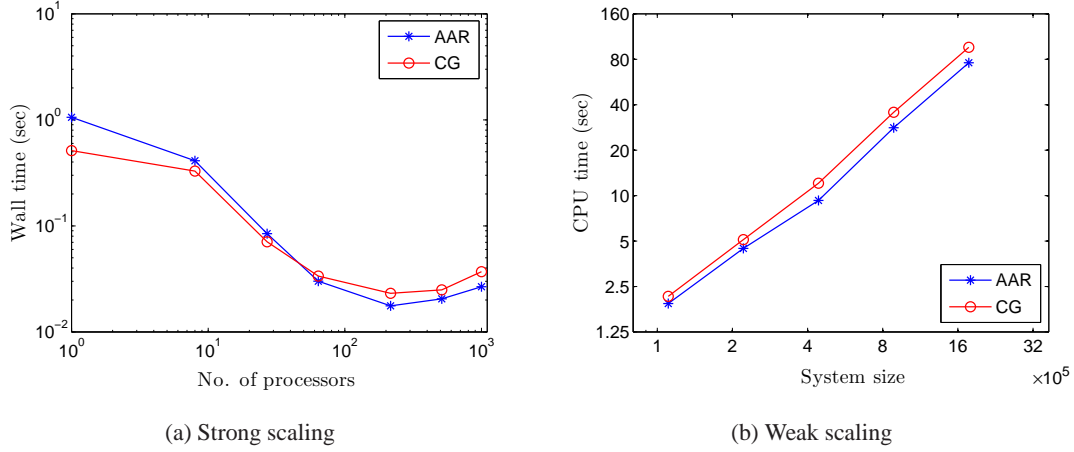


Figure 4: Strong and weak scaling of AAR and CG for the periodic Poisson equation with ILU(0) block Jacobi preconditioning. In strong scaling, the minimum wall time taken by AAR is 1.31 times smaller than CG. In weak scaling, the CPU time taken by AAR and CG scales with system size as $\mathcal{O}(N^{1.33})$ and $\mathcal{O}(N^{1.38})$, respectively.

the serial and parallel settings.³

3.2. Poisson equation

Next, we study the performance of AAR relative to CG for solving the linear system arising from the discretization of the Poisson problem (Eq. 7b). We again consider a $3 \times 3 \times 3$ Al supercell based on a FCC unit cell with lattice constant 7.78 Bohr, with atoms randomly displaced from ideal positions, a mesh-size of $h = 0.486$ Bohr, and ILU(0) block Jacobi preconditioning. In Fig. 4a, we plot the wall time taken by AAR and CG on 1, 8, 27, 64, 216, 512, and 1000 computational cores. At small core counts, CG demonstrates better performance than AAR by virtue of requiring fewer iterations to achieve convergence. However, as the number of cores is increased, the performance of AAR relative to CG improves, with the minimum wall time taken by AAR being a factor of 1.31 smaller than CG by virtue of the lesser global communication required by AAR.

We now perform a weak scaling study by periodically replicating the above $3 \times 3 \times 3$ system along one direction by factors of 1, 2, 4, 8, and 16, i.e., we generate $3 \times 3 \times 3$, $6 \times 3 \times 3$, $12 \times 3 \times 3$, $24 \times 3 \times 3$, and $48 \times 3 \times 3$ supercells. Correspondingly, we choose 64, 128, 256, 512, and 1024 computational cores. Again, we select $h = 0.486$ Bohr and employ ILU(0) block Jacobi preconditioning. In Fig. 4b, we plot the CPU time taken by AAR and CG for the resulting systems, from which we obtain the weak scaling with system size to be $\mathcal{O}(N^{1.33})$ and $\mathcal{O}(N^{1.38})$, respectively. As before, even though the number of iterations does not vary with system size, the increasing cost associated with global communications results in superlinear scaling for both approaches. Therefore, the performance of AAR relative to CG is expected to further improve as core counts are increased, a result which we verify next.

³The upturns in strong scaling plots at largest core counts arise due to insufficient computational work per core relative to local inter-processor communications when the chosen computation is spread beyond a certain number of cores. This indicates the strong scaling limit of the current implementation for the chosen problem size.

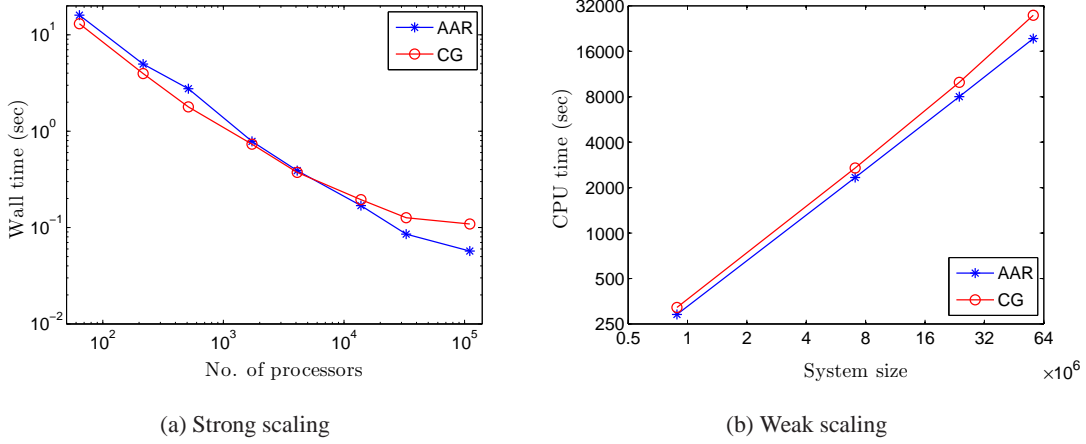


Figure 5: Strong and weak scaling of AAR and CG for the periodic Poisson problem with Jacobi preconditioning. In strong scaling, the minimum wall time taken by AAR is 1.91 times smaller than CG. In weak scaling, the CPU time taken by AAR and CG scales with system size as $\mathcal{O}(N^{1.01})$ and $\mathcal{O}(N^{1.07})$, respectively.

To assess the efficiency and scaling of AAR in larger-scale parallel calculations, up to 110,592 cores, we consider strong and weak scaling on the Vulcan IBM BG/Q machine at the Lawrence Livermore National Laboratory. For parallel scalability, we choose a simple Jacobi preconditioner. For the strong scaling study, we choose a $12 \times 12 \times 12$ Al supercell with atoms randomly displaced, a mesh-size of $h = 0.486$ Bohr, and a maximum of 110,592 computational cores. For the weak scaling study, with $h = 0.486$ Bohr, we go from $6 \times 6 \times 6$ supercell on 1728 processors to $24 \times 24 \times 24$ supercell on 110,592 processors, with atomic displacements, electron density, and pseudocharge density periodically repeated from the $6 \times 6 \times 6$ system. We present the results obtained in Fig. 5. We find that the minimum wall time achieved by AAR within the number of cores available for this study is 1.91 times smaller than that achieved by CG. In addition, the weak scaling with system size for AAR and CG are $\mathcal{O}(N^{1.01})$ and $\mathcal{O}(N^{1.07})$, respectively. This demonstrates again the increased advantage of AAR over current state-of-the-art Krylov solvers in parallel computations as the number of processors is increased.

4. Concluding remarks

We generalized the recently proposed Alternating Anderson-Jacobi (AAJ) method to include preconditioning, and demonstrated its efficiency and scaling in the solution of large, sparse linear systems on parallel computers. The resulting preconditioned Alternating Anderson-Richardson (AAR) method reduces to the AAJ method for the particular choice of preconditioner, $\mathbf{M}^{-1} = \text{diag}(\mathbf{A})^{-1}$. The AAR method employs Anderson extrapolation at periodic intervals within a preconditioned Richardson iteration to accelerate convergence. In applications to Helmholtz and Poisson equations, we showed that the strong and weak parallel scaling of AAR is superior to both Generalized Minimal Residual (GMRES) and Conjugate Gradient (CG) methods, using the same preconditioning, in large-scale parallel calculations employing up to 110,592 computational cores. Moreover, we find that the minimum time to solution for AAR is up to a factor of two smaller than both GMRES and CG. Our findings suggest that the AAR method provides an efficient and scalable alternative to current state-of-the-art preconditioned Krylov solvers for the solution of large, sparse

linear systems on high performance computing platforms, with increasing advantage as the number of processors is increased. Additional mathematical analysis which provides further insights into the performance of the AAR method and therefore enables the development of effective preconditioners tailored to it will enable still larger-scale applications, and so constitutes a potentially fruitful subject for future research.

5. Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. We gratefully acknowledge support from the Laboratory Directed Research and Development Program. P.S. and P.P. also gratefully acknowledge the support of National Science Foundation under Grant Number 1333500.

References

- [1] Y. Saad, M. H. Schultz, *SIAM Journal on Scientific and Statistical Computing* 7 (1986) 856–869.
- [2] J. R. Shewchuk, *An introduction to the conjugate gradient method without the agonizing pain*, 1994.
- [3] F. Shimojo, R. K. Kalia, A. Nakano, P. Vashishta, *Computer Physics Communications* 167 (2005) 151–164.
- [4] Y. Saad, *Iterative Methods for Sparse Linear Systems* (2nd ed), SIAM, 2003.
- [5] R. Barrett, M. W. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. Van der Vorst, *Templates for the solution of linear systems: building blocks for iterative methods*, volume 43, SIAM, 1994.
- [6] X. I. Yang, R. Mittal, *Journal of Computational Physics* 274 (2014) 695–708.
- [7] P. P. Pratapa, P. Suryanarayana, J. E. Pask, *Journal of Computational Physics* 306 (2016) 43–54.
- [8] D. G. Anderson, *Journal of the Association for Computing Machinery* 12 (1965) 547–560.
- [9] A. S. Banerjee, P. Suryanarayana, J. E. Pask, *Chemical Physics Letters* 647 (2016) 31–35.
- [10] N. Choly, E. Kaxiras, *Solid State Communications* 121 (2002) 281–286.
- [11] S. Ghosh, P. Suryanarayana, *Journal of Computational Physics* 307 (2016) 634–652.
- [12] P. Suryanarayana, D. Phanish, *Journal of Computational Physics* 275 (2014) 524–538.
- [13] J. E. Pask, P. A. Sterne, *Phys. Rev. B* 71 (2005) 113101.
- [14] P. Suryanarayana, K. Bhattacharya, M. Ortiz, *Journal of the Mechanics and Physics of Solids* 61 (2013) 38–60.
- [15] J. E. Pask, N. Sukumar, S. E. Mousavi, *International Journal for Multiscale Computational Engineering* 10 (2012) 83–99.
- [16] S. Ghosh, P. Suryanarayana, *arXiv preprint arXiv:1603.04339* (2016).

- [17] P. Suryanarayana, V. Gavini, T. Blesgen, K. Bhattacharya, M. Ortiz, *Journal of the Mechanics and Physics of Solids* 58 (2010) 256–280.
- [18] P. Suryanarayana, K. Bhattacharya, M. Ortiz, *Journal of Computational Physics* 230 (2011) 5226–5238.
- [19] S. Ghosh, P. Suryanarayana, arXiv preprint arXiv:1603.04334 (2016).
- [20] S. Balay, J. Brown, , K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, H. Zhang, *PETSc Users Manual*, Technical Report ANL-95/11 - Revision 3.4, Argonne National Laboratory, 2013.
- [21] S. Balay, W. D. Gropp, L. C. McInnes, B. F. Smith, in: E. Arge, A. M. Bruaset, H. P. Langtangen (Eds.), *Modern Software Tools in Scientific Computing*, Birkhäuser Press, 1997, pp. 163–202.
- [22] A. J. Laub, *Matrix Analysis for Scientists and Engineers*, SIAM, 2005.